Design of a Secure and Decentralized Location Service for Agent Platforms

Overeinder, B.J.; Oey, M.A.; Timmer, R.J.; van Schouwen, R.M.; Rozendaal, E.; Brazier, F.M.

published in
Proceedings of the Sixth International Workshop on Agents and Peer-to-Peer Computing (AP2PC 2007)
2007

document version
Publisher's PDF, also known as Version of record

Design of a Secure and Decentralized Location Service for Agent Platforms

B.J. Overeinder[1], M.A. Oey[1], R.J. Timmer[1], R. van Schouwen[1], E. Rozendaal[2], and F.M.T. Brazier[1]

[1] IIDS Group, Department of Computer Science, Faculty of Sciences, Vrije Universiteit Amsterdam, de Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{bjo,michel,rjtimmer,reinout,frances}@cs.vu.nl
[2] NLnet Labs, Kruislaan 419, 1098 VA Amsterdam, The Netherlands

Abstract. Agent platforms designed for Internet-scale, open networks need scalable and secure location services for agents and services. The location service based on the Fonkey public key distribution infrastructure presented in this paper has been designed and implemented for this purpose. It is scalable in the total number of published identifier–contact address pairs, the number of updates/changes, and the number of agent platforms publishing and requesting contact addresses. This system also supports a signing mechanism to authenticate the publisher of an identifier–contact address pair. Experimental results show that the current implementation based on the Bunshin/Free Pastry overlay network exhibits good scaling behavior.

1 Introduction

Scalable location services are essential in distributed systems and, in particular, for multi-agent systems. The Domain Name System (DNS) is a very successful realization of a location service that resolves symbolic names to contact addresses (IP addresses). DNSSEC (Secure DNS) has been designed to support authentication preventing spoofing and man-in-the-middle attacks [1].

Both DNS and DNSSEC, however, are not designed to deal with highly dynamic entities such as mobile agents. The dynamic nature of mobile agents in Internet-scale, open network systems requires a different type of approach for registering, deregistering, and retrieving location information. Scalability and integrity are of utmost importance as (up-to-date) agent location information is a prerequisite to successful agent mobility.

This paper presents the design of a scalable and secure location service based on the Fonkey system. Fonkey is an infrastructure for global public key (optionally with a payload) distribution. The payload in our location service is agent location information, optionally signed by other public keys.

Section 2 discusses current technologies used in name resolution, such as DNS and LDAP, and location services for (mobile) agent systems. Section 3 presents the design of the Fonkey-based location service. Section 4 discusses security issues, and Sect. 5 reports on experimental scalability results. Finally, Sect. 6 concludes the paper with a discussion on future work.

2 Background

The most widely used location service on the Internet today is Domain Name System (DNS) [2]. DNS defines a hierarchical name space used to map computer host names and other resources to IP addresses. DNS is a distributed database that allows for scalable operation by distributing the hierarchical name space over many servers, each responsible for a specific part of the name space. DNSSEC [3] is a security extension for DNS that cryptographically ensures DNS data is not forged or altered. The DNSSEC extensions provide data integrity and authentication through the use of digital signatures.

The Lightweight Directory Access Protocol (LDAP) [4] is a directory service that can also be used as a location service. The LDAP protocol is designed to provide access to directories supporting the X.500 models. The directory access protocol provides both read and update access. Update access requires secure authentication. LDAP organizes data in a hierarchy using the Distinguished Name (DN). Like DNS, LDAP allows distribution over multiple servers based on the hierarchical name.

Van Steen *et al.* [5] present a location service for mobile objects in a worldwide system named Globe. The mobile object location service strictly separates an object's name from the location on which it resides. This is done by binding an object's name to a location-independent object handle, which, in turn, is mapped to the location where the object resides. The location service is organized as a distributed search tree. To achieve scalability of the hierarchical location service, where potentially high-level nodes may become a bottleneck, location information is distributed such that the load is evenly balanced, while at the same time exploiting locality [6]. Name and location service NLS extends the Globe location service by using prefixes to aggregate location information and using a two-layer architecture with fat-trees at the global layer [7].

Locating mobile *agents* in worldwide distributed systems is also specifically addressed in a number of research papers. Di Stefano and Santoro [8] propose a naming scheme and location protocol with intended general validity for mobile agents able to effectively meet all the typical requirements of mobile agent environments and, thus, straightforward to integrate into different platforms. Functionality for authentication of hosts publishing location updates and information integrity is not, however, considered in their system. Roth and Peters [9] propose a global tracking service for mobile agents, designed to scale to the Internet, and to be secure. Their location information load sharing approach is similar to the Globe location service, but does not presuppose coherent mobile agent migration patterns to achieve scalability. The protocols presented by Roth and Peters have a number of advantageous security properties, in particular, malicious location updates by unauthorized hosts are prevented. A scalable hash-based mobile agent location mechanism is proposed by Kastidou *et al.* [10]. A mobile agent tracking mechanism based on hashing is presented, and dynamic rehashing is supported to allow the system to adapt to variable workloads.

The research reported in this paper has defined design goals similar to the global tracking system presented in [9], namely a scalable and secure location service for mobile agents. However, the location service system presented in this paper includes data integrity verification.

3 A Fonkey-Based Location Service

The Fonkey system was designed by NLnet Labs [11] for the distribution of public keys with the option to include additional data, possibly signed by other public keys. The combined distribution of public keys and signed data allows for authorized updates and integrity of information.

The Fonkey-based location service extends this system with registration, deregistration, and retrieval of agent location information, implemented in a client library described in Sect. 3.2.

3.1 Overview of the Fonkey Infrastructure

This section first describes how Fonkey works: Fonkey's basic functionality, the data structures (i.e., types of packages) involved, and the payload. Next, the details are presented how the different types of packages are located.

Fonkey Basic Operation Although Fonkey resembles Public Key Infrastructures (PKI), it only provides a subset of the functionality normally found in PKIs. The design of Fonkey is intended to provide a common layer implementing mechanisms for higher and more complex software architectures such as PKIs or DNSSEC.

In the Fonkey system, principals[1] generate a public/private key pair and publish the public key together with reference information. The public key becomes a means of identification. There is, however, no cryptographic link between the key and the principal publishing the key. Keys in Fonkey can, however, be signed by other keys. The level of trust in a key can be (recursively) determined by checking the signatures belonging to that key.

Fonkey does not keep information indefinitely: information in Fonkey expires. Once information expires, it is removed from the Fonkey system and must be re-published into Fonkey. The automatic expiration of data is done to avoid having outdated packages stay in Fonkey permanently, and provides a simple garbage collection mechanism.

Fonkey Package Structures The three basic concepts involved in publishing and retrieving public keys are: keys, named data, and signatures. These three concepts directly translate into three data structures supported by Fonkey: Key Package, Named Package, and Signature Package. This section presents these package structures and how they are used by Fonkey operations.

Basic Package Structure. The three different package types in Fonkey have the following common elements:

[1] A principal is an entity whose identity can be authenticated.

Package-id	A unique identifier to identify this package.
Type	Type of package is either `Key`, `Named`, or `Signature`.
Public Key	The public key component of the key pair used to sign the package. This key can be used to verify the integrity of the package.
Version	A strictly increasing version number to ensure older packages do not accidentally overwrite newer packages.
Properties	A set of name/value pairs.
Payload	Application specific payload.
Signature	The signature used to ensure the integrity of the package.

Key Package. The structure of a Key Package is identical to the basic package.

Named Package. The structure of a Named Package is equivalent to the basic package extended with an extra field, the Name field:

Name	The name of this package. The name can be an arbitrary string. The name is used to locate this package. The name could be a public key, an e-mail address, a host name, etc.

Signature Package. The structure of a Signature Package is that of the basic structure extended with two additional fields, the Subject and References fields:

Subject	The unique package-id of the package that is signed by this signature package.
References	The parts of the subject package being signed. For each part a SHA-1 hash is stored.

Payload Each package in Fonkey holds payload information (limited in size). The client application is responsible for processing the payload.

Publishing Packages Principals publish packages in Fonkey so that other principals can retrieve them. Published packages never 'overwrite' existing packages (version numbers must always be different). Old packages simply 'expire'. Different principals can publish the same information by each publishing a package (e.g., a Named Package). In that case, a principal retrieving that information will receive both packages, signed by different principals. The trustworthiness of the information in the packages then depends on the amount of trust it has in the signing principals.

Retrieving Packages One of the most important features of Fonkey is package retrieval. The identifiers with which the three types of packages can be retrieved differ. All packages can retrieved based on their `Package-id`. In addition:

1. Named Packages can be retrieved based on their `Name` value.
2. Key Packages can be retrieved based on their of `Public Key` value.
3. Signature Packages can be retrieved based on their `Subject` value.

Retrieving packages from Fonkey will return all matching packages (possibly signed by different principals).

3.2 Location Service Client

This section describes the way in which Fonkey has been extended to implement a location service. As Fonkey itself was not specifically designed to be an agent location service, a mapping is needed between location service and Fonkey operations. A client library has been designed to this purpose.

Named Packages are the primary data structure used by location and name services. The `Name` of a Named Package can be used to look up the associated public key and payload. For an agent location service, the name is the agent identifier. The naming scheme is application specific. For example, e-mail addresses or domain names can follow a hierarchical naming scheme, but for open, peer-to-peer systems such as agent platforms, a flat naming scheme is more appropriate.

Fonkey can be used as the basis for a location service in the following way:

- Each agent platform publishes its public key in a Key Package.
- The location of an agent is published in a Named Package: the Name field of the Named Package is the agent's unique identifier, and the package's Payload is the agent's location. Each platform is responsible for publishing and signing the location information of all agents running on that platform.
- Agent platforms can sign Key Packages of other agent platforms with Signature Packages. The Signature Packages can create a "Web of Trust" for others querying the location service.

Below the main functionality of a location service is discussed.

Registering an Agent. When an agent is injected into the system its location needs to be registered with Fonkey. This is done by creating a Named Package. The name is the agent's unique identifier, and the contents of the package contain the agent's location. The package is signed and published by the agent platform instance in which the agent is located.

Agent Lookup. Agent lookup is done by searching Fonkey for a Named Package with the name being the agent's unique identifier:

- If no packages are found the agent does not exist (anymore).
- If one or more packages are found, the platform filters the packages by only looking at packages published by known and trusted platforms. The package with the highest version number (e.g., timestamps could be used as version numbers) indicates the current location of the agent.

Agent Migration. Agent migration is the most complicated scenario: care must be taken to ensure the agent is not accidentally "dropped" or duplicated, for example, when one of the locations crashes or network connectivity is lost. Another important issue is to correctly update an agent's location in Fonkey.

The basic agent migration procedure is as follows, given an agent A, and locations X and Y.

- Agent A, running on location X, indicates its wish to migrate to location Y.

– Location X contacts location Y and transfers agent A.
– Location Y acknowledges to location X that agent A has been received.
– Location X stops republishing location information packages for agent A, but maintains a forwarding pointer for agent A to location Y in case other agents try to contact agent A on the old location.
– Location Y publishes a new Fonkey Named Package indicating that agent A is located at location Y. As this package will have a higher version number than the previous package published by location X, this marks location Y as the current location of agent A.

4 Security

Authenticity, integrity and non-repudiation are important aspects of security [12] that have been addressed in the design of the Fonkey location service. Confidentiality of communication between agent platforms and a Fonkey location service is not an issue, nor is confidentiality of the information stored.

All packages published in Fonkey are signed using the private key of the publishing entity. The validity of the signature, and the integrity of the package are checked by Fonkey, and refused if any part is not valid. To prevent malicious agent platforms from publishing false agent locations, verification is needed to determine whether a Named Package contains the currently valid agent location. Validity of the locations is addressed by using a trust model that determines the trustworthiness of the location information.

Fonkey itself, however, does not implement a trust model, but provides the mechanisms for trust verification operations. In this paper a simple trust model is implemented: each Fonkey location service is configured with a list of other "trusted" location service identifiers, under control of the administrator of the local agent platform on which the location service runs. Only packages signed by one of these trusted locations are trusted. With Fonkey's Signature Packages, this simple trust model can be extended to a more distributed and scalable model where trust is a transitive relationship.

Non-repudiation is not guaranteed in all cases. Although a malicious Fonkey host cannot alter the information published (due to the publisher's signature), the Fonkey server can deny ever having received a package, or return an older version of the same package. In this paper, however, all Fonkey servers are assumed to be trusted.

5 Scalability Experiments

A secure and decentralized location service based on the design described above has been implemented and evaluated. Section 5.1 describes the architecture implementation, and Sect. 5.2 evaluates the service's performance.

5.1 Implementation

Figure 1 shows the general architecture of the decentralized implementation. The Fonkey infrastructure is a distributed data store based on the Bunshin DHT implementation [13]. Bunshin uses the FreePastry [14] overlay network for routing messages. All

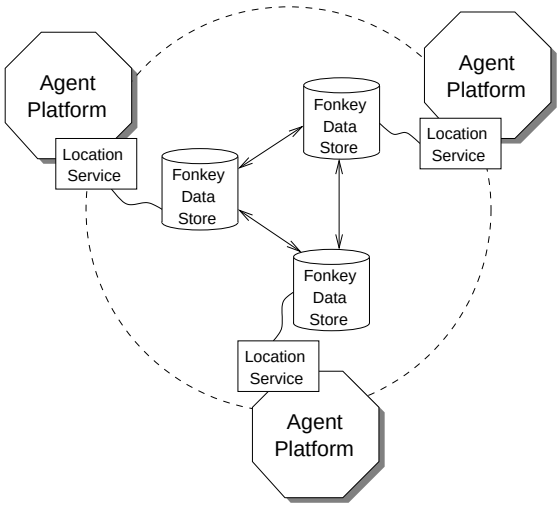

Fig. 1. Fonkey location service abstract architecture.

components of the Fonkey prototype are implemented in the Java programming language.

The Fonkey infrastructure is designed to be used as a component in the solution of various problems (applications) in which authentication and data integrity is an issue of concern. Figure 1 shows how an agent platform can specifically use the Fonkey infrastructure to implement a secure location service. The agent platform uses a specific location service client library that interacts with the Fonkey server side (the Fonkey data store). The location service client library abstracts from the Fonkey primitive operations and provides functionality for identifier lookup and (de-)registration. The location service client library also checks the signature of the signed data in Fonkey packages, and establishes trust relationships between public keys (using Signature Packages).

Agent platforms are typically distributed systems and can consist of multiple hosts/-resources (e.g., JADE [15] and AgentScape [16]). An agent platform can connect to any Fonkey server to hook up into the Fonkey infrastructure. Preferably, each platform is connected with a local Fonkey server for lower network latencies to the Fonkey server and to allow for more effective DHT optimizations (replication and caching).

Registering an agent requires the following actions:

1. create a new Named Package for the agent with the location information.
2. retrieve (*get*) a list of all existing Named Packages (possibly signed by different platforms) for the corresponding agent from the DHT.
3. add the newly made Package to this list and *put* the list back into the DHT.

Note that a register operation actually requires a get and a put operation. Looking up an agent's location requires the following actions:

1. *get* a list of all Named Packages for the corresponding agent from the DHT.
2. for each package *get* the Key Package of the platform that signed it.

3. for each Key Package *get* all Signature Packages signing it.

Steps 2 and 3 must be repeated until a Signature Package published by a trusted platform is found.

5.2 Experiments

To evaluate the design and implementation of the decentralized Fonkey location service described above, a series of experiments have been implemented and executed.

The experiments are performed on the DAS-3 cluster, a high-performance cluster with AMD Opteron dual-core 2.4 GHz processors and Gigabit/s Ethernet network.[2] In the experimental setup, agent platforms are synthesized by processes generating identifier register and lookup operations using the location service client library. A synthesized agent platform and associated local Fonkey server are exclusively co-allocated to a node in the cluster (only one pair per node).

The experiments measure the amount of time needed to complete register and lookup operations for increasing numbers of Fonkey servers and agent platform clients. For experimental testing purposes, the Fonkey DHT data stores are pre-filled with identifier–location information, such that identifier lookup operations are always successful. The Fonkey location service register and lookup operations are evaluated in three different settings: (i) register only; (ii) lookup only; and (iii) register and lookup mix (with ratio 20:80).

In the first series of experiments, the DHT overlay network is flooded with register and/or lookup requests for all three different settings. The synthesized agent platforms generate a stream of register, lookup, or a mix of register/lookup requests; and as soon as a request has been completed, the next request is made to the local Fonkey server.

Figure 2 shows the results for the three different register/lookup settings. Given that each single node runs a Fonkey server and a synthesized agent platform, both the number of servers and number of clients are increased equally with the number of nodes from 1 to 64. Thus, not only the number of DHT stores that contain the distributed location information increase, but also the number of register/lookup requests grow for each data point in Fig. 2. The time for retrieving data items in a DHT data store grows logarithmically with the size of the overlay network, *and* the amount of traffic grows by the increasing number of agent platform clients issuing register/lookup requests. The results are the average of three experiments with an error of less than 10%.

The lookup results in Fig. 2 show how the completion time for a single lookup operation increases with the number of nodes. In a distributed data store such as a DHT, more nodes implies a smaller percentage of local package retrieval and a larger percentage of remote interaction. The curve shows very reasonable scaling behaviour up to 32 nodes. The results for 64 nodes indicate (possibly) another, more linear scaling trend. The register results show an almost constant curve.

The register completion times for 1 to 4 are higher than for lookup, while from 8 and higher the times are lower. Register times are higher for smaller number of nodes because of two reasons: the high fixed cost of signing packages and the actual local store

[2] `http://www.cs.vu.nl/das3/`

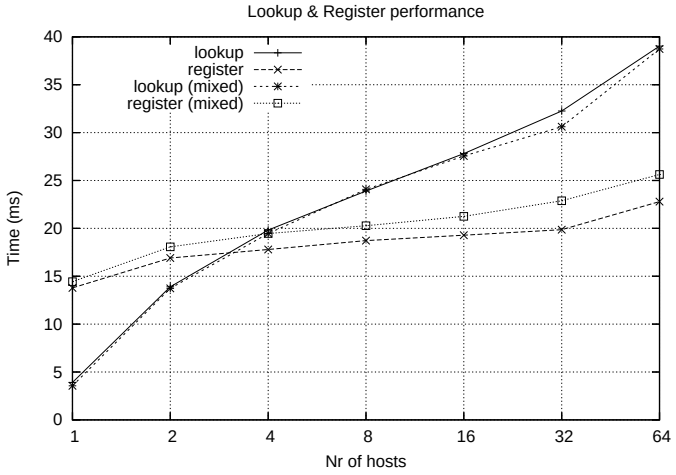

Fig. 2. Completion times of Fonkey location services register and lookup operations.

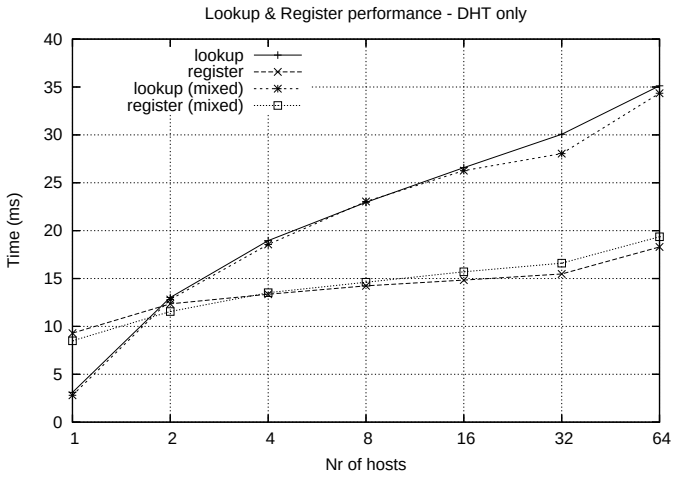

Fig. 3. Completion times of the plain DHT infrastructure put and get operations.

operation. Signing a package takes about 4 ms compared to only 1 ms for verifying a signature. Furthermore, recall that a register operation actually does a *get* operation on the DHT first (see Sect. 5.1).

For larger numbers of nodes register operations are faster than lookup operations because lookup operations are synchronous while register operations are mostly asynchronous. A lookup operation is not complete until the requested package has been retrieved from the (possible) distant DHT data store. In our experiment, the situation is even worse: a lookup operation actually has to retrieve two packages from the DHT (see

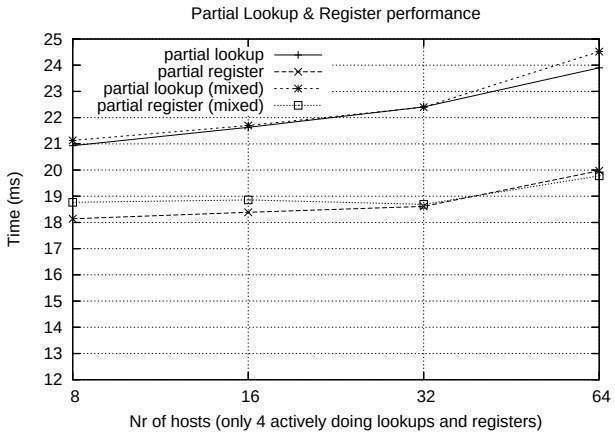

Fig. 4. Completion time results for the Fonkey location service with four active clients.

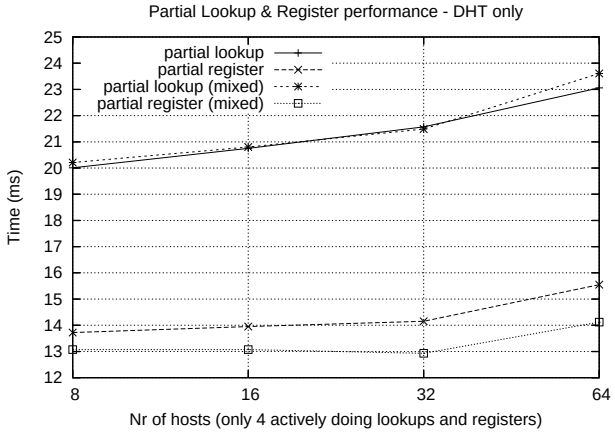

Fig. 5. Completion time results for the plain DHT infrastructure with four active clients.

Sect. 5.1). In contrast, to a client, a register operation is complete as soon as the package has been accepted by the local Fonkey server, while the underlying overlay network routes the package to the destination DHT data store. Therefore, the more nodes a DHT has, the more the expensive local store operation can be performed concurrently with the next register operation.

The results of lookup (mixed) and register (mixed) operations are quite similar to their individual results. Only for the register operation, can a small increase in completion time be seen, due to interference with the synchronous lookup operation.

To complement the Fonkey location service performance evaluation, similar experiments have been conducted with the underlying DHT only. The results in Fig. 3 show that the Fonkey overhead is relatively small, and its scaling behaviour is mainly due to

the scaling characteristics of the underlying Bunshin DHT implementation. The change in scaling behaviour for the 64 node experiments needs further study in the underlying Bunshin DHT and/or FreePastry overlay network implementation.

A second series of experiments was conducted to study the Fonkey location service performance in a scenario where only four agents are concurrently active (generating fewer requests and traffic than the first scenario). Figure 4 shows the results of the experiments from 8 to 64 nodes. The results for 8 and 16 nodes are close to the results in Fig. 2. For 32 and 64 nodes, the results are much better: the completion time for lookup operation dropped 35%. Because the overlay network is not flooded with messages, the synchronous lookup operation completes in less time. The results for plain DHT experiments in Fig. 5 show similar behaviour.

6 Discussion and Future Work

This paper presents the design of a scalable and secure location service for agent platforms. The location service is based on the Fonkey public key distribution infrastructure. This public key distribution system and the publishing of signed data associated with public keys allows for the implementation of a secure location service where publication of location information can be authenticated and the integrity of the information can be verified.

The design of the decentralized Fonkey infrastructure based on distributed hash tables relates with previous work on DHT-based location services as presented by Stoica *et al.* [17] and Rowstron *et al.* [14]. The current implementation of Fonkey uses the Bunshin/FreePastry overlay network substrate to implement its distributed hash table package stores. The Fonkey infrastructure adds operations for publishing and retrieving public key packages with associated data. The packages can be searched for in a number of ways, facilitating the implementation of a number of applications relying on public key management and operations. The secure location service described in this paper is implemented as a client library for the Fonkey system.

From the experimental results, one can conclude that the Fonkey infrastructure based on DHT overlay networks scales. Analysis of the time spent per register/lookup operation indicates that for large number of clients, the total latency of the operations is determined by the DHT overlay network latency. The results presented are not yet optimized for data replication and caching.

In the near future the results from the experiments and planned experiments with Fonkey in real-world setting with a multi-agent middleware, will be compared with an alternative approach to distributed directory services based on agent clustering [18]. With agent-based clustering, agents organize themselves in clusters with similar characteristics. The resulting clustering graph, or overlay network, can be used to find information efficiently and in an associative manner.

Acknowledgments

This research is supported by the NLnet Foundation, `http://www.nlnet.nl/`. The authors thank Miek Gieben and Ted Lindgreen from NLnet Labs for their valuable contributions.

References

1. Atkins, D., Austein, R.: Threat analysis of the domain name system. IETF Internet-Draft draft-ietf-dnsext-dns-threats-03.txt (June 2003)
2. Mockapetris, P.: Domain names – Concepts and facilities. IETF RFC 1034 (November 1987)
3. Arends, R., Austein, R., Larson, M., Massey, D., Rose, S.: DNS security introduction and requirements. IETF RFC 4033 (March 2005)
4. Wahl, M., Howes, T., Kille, S.: Lightweight directory access protocol (v3). IETF RFC 2251 (December 1997)
5. van Steen, M., Hauck, F., Homburg, P., Tanenbaum, A.: Locating objects in wide-area systems. IEEE Communications Magazine **36**(1) (January 1998) 104–109
6. van Steen, M., Ballintijn, G.: Achieving scalability in hierarchical location services. In: Proceedings of the 26th International Computer Software and Applications Conference (COMPSAC 2002), Oxford, UK (August 2002) 899–906
7. Hu, Y., Rodney, D., Druschel, P.: Design and scalability of NLS, a scalable naming and location service. In: Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 2002), New York, NY (June 2002)
8. Di Stefano, A., Santoro, C.: Locating mobile agents in a wide distributed environment. IEEE Transactions on Parallel and Distributed Systems **13**(8) (August 2002) 844–864
9. Roth, V., Peters, J.: A scalable and secure global tracking service for mobile agents. In: Mobile Agents 2001. Volume 2240 of Lecture Notes on Computer Science. Springer-Verlag, Berlin, Germany (2001) 169–181
10. Kastidou, G., Pitoura, E., Samaras, G.: A scalable hash-based mobile agent location mechanism. In: Proceedings 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW'03), Providence, Rhode Island (May 2003) 472–477
11. NLnet Labs: Fonkey project. `http://www.nlnetlabs.nl/fonkey/`
12. Menezes, A., van Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, FL (1997)
13. Mondéjar, R., Pairot, C., García, P.: Bunshin: DHT replication and caching. `http://planet.urv.es/bunshin/`
14. Rowstron, A., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware 2001. Volume 2218 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany (2001) 329–350
15. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with a FIPA-compliant agent framework. Software: Practice and Experience **31**(2) (June 2001) 103–128
16. Overeinder, B., Brazier, F.: Scalable middleware environment for agent-based Internet applications. In: Proceedings of the Workshop on State-of-the-Art in Scientific Computing (PARA'04), Copenhagen, Denmark (June 2004) 675–679
17. Stoica, I., Morris, R., Liben-Nowell, D., Karger, D.R., Kaashoek, F., Dabek, F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup protocol for Internet applications. IEEE/ACM Transactions on Networking **11**(1) (February 2003) 17–32
18. Ogston, E., van Steen, M., Brazier, F.: Group formation among decentralized autonomous agents. Applied Artificial Intelligence **18**(9–10) (October–December 2004) 953–970